Centralised High-Level Planning for a Robot Fleet

Centralised High-Level Planning for a Robot Fleet

Matthew Crosby and **Ronald P. A. Petrick**
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
m.crosby@ed.ac.uk, rpetrick@inf.ed.ac.uk

Abstract

This paper describes an application of automated planning to the problem of computing and distributing goals, and initial action sequences (plans), to a fleet of factory robots using a centralised controller. The initial plans must be designed and distributed such that the total expected execution time is reduced. Whilst we found that existing decomposition-based approaches could find plans in the required time frames for this task, the returned plans were of low quality in terms of agent distribution. As a result, this paper introduces new methods for decompositional planning that output more balanced plans, whilst still retaining enough of the speed benefits of the original approach to meet our time constraints.

Introduction

This paper describes an application of automated planning to the real-world problem of assigning goals and actions to a fleet of mobile robots in a factory environment. In this domain, robots must navigate a factory floor, each collecting a set of parts to be used in a manufacturing process. Due to the limitations of the environment. and the robots themselves, they must follow a preset path and cannot overtake one another as they move through the factory. A centralised controlled, called the *mission planner*, is tasked with creating initial high-level plans for the robots in the fleet. The planner has information about the capabilities of each robot and the goals for the fleet as a whole, and the plans it generates are responsible for assigning goals, setting partial-order execution constraints, and providing initial plans to the robots.

While it might be argued that a domain-specific (non-planning) technique, tailored to the exact nature of the problem, could provide an efficient solution in this environment, technology reuse remains an important factor in modern industrial settings. As such, general purpose domain-independent planning offers an approach that can be utilised even as the constraints of the problem and domain evolve over time, while also providing certain desirable properties in an industrial setting: 1) the technology can be reused with only minor modifications to the input (PDDL files) when robot capabilities and the environment change, 2) the system can be upgraded as more advanced (multiagent planning) algorithms become available, and 3) the outputs (plans) are produced in a standard format based on well-defined inputs (PDDL) that can subsequently be processed by other systems (such as execution monitors or other task planners).

Although planning offers a promising solution, the nature of the domain itself presents certain technical challenges, even for state-of-the-art approaches. Initial experiments in this domain showed that standard planners struggled with this type of problem, especially as the number of robots increased. For instance, temporal planners could find plans with low makespan, but could only cope with small 'toy' instances of the domain (2 robots, 4 goals). Standard single-agent planners could solve slightly larger problems (2 robots, 6 goals), but still not scale to the sizes needed (10 robots, 10 goals). The planners struggled especially with the number of robots in the domain.

One positive result from these initial experiments came in the form of multiagent decomposition techniques. Even though the domain is not loosely coupled (all actions are public and can affect what other agents can do), it is still amenable to decomposition-based approaches. In particular, the multiagent decomposition-based planner ADP (Crosby, Rovatsos, and Petrick 2013) was able to decompose the domain automatically, separating out the robots and finding solutions to the full size problem in under a second. The automatic decomposition also meant that a flat PDDL encoding could be used, without any additional special markup to specify agents. Unfortunately, the solutions did not distribute the plans amongst the agents in any meaningful way.

This paper focuses on our particular application domain and the problem of creating a new version of ADP, we have named ADBP (Agent Decomposition Balancing Planner), that builds on the partial success of the basic ADP technique to generate higher-quality (i.e., better balanced) plans. The aim of this work is to enable ADBP to still solve our testing domains within a reasonable time frame (10 seconds), whilst also outputting plans with reduced makespan compared to ADP. To explore the limits and further possible applications of this work, we also tested ADBP on multiagent planning domains taken form the International Planning Competition that ADP is known to perform well on.

Background

Planning has a long association with robotics research, with the application of modern planning techniques to robotics problems gaining popularity in recent years. However, the general use of planning has often centred around robot-level task planning which, while abstracted from low-level robot control, still needs to deal with contingencies, sensing actions, and other robot-level considerations. The work discussed in this paper considers a problem that is abstracted even further; while we avoid some of the complexities that arise in typical robotics planning problems, we also introduce new complications (e.g., multiple robots). As such, the problem we address is more akin to that of classical planning but with real-world constraints that cannot easily be avoided.

The domain encoding we use in this paper is motivated by the abstraction of robot-level motor programs into *robot skills* (Bøgh et al. 2012). Robot skills are designed to bridge the gap between robot-level control operations and planning-level actions, by providing a structured representation of both the requirements (preconditions) and expected outcomes (effects) of a robot-level action, encapsulated together with a set of methods for verifying these conditions in the real world through sensing. Our encoding of the domain abstracts away the verification components of the skills structure to encode just the preconditions and effects.

Initial testing in this domain focused on a temporal planning encoding using the capabilities introduced in PDDL 2.1 (Fox and Long 2003). This enabled the use of numeric fluents to simplify the encoding, resulting in low makespan plans in which robot actions were scheduled concurrently. While temporal planning has been previously used in multiagent settings, it has not been prevalent since (Brenner 2003), and initial testing with POPF2 (Coles et al. 2010), a forward chaining partial-order planner that was the runner up in the temporal track of the 2011 IPC (Coles et al. 2012), established it could not scale to the full version of our domain.

Other tests were also performed using the popular planners LAMA (Richter and Westphal 2010) and FF (Hoffmann and Nebel 2001) on a standard STRIPS-style PDDL encoding of the domain (Fikes and Nilsson 1971). While these planners managed to solve larger problems, they broke down as the number of robots increased.

Given the centralised and cooperative nature of our application, and motivated by the results of the initial testing, this work instead focuses on multiagent techniques that attempted to exploit the underlying structure inherent in multiagent domains (Brafman and Domshlak 2008), while avoiding approaches that deal with decentralised planning, strategic elements or privacy concerns. The particular planner selected for this application, ADP (Crosby, Rovatsos, and Petrick 2013), performs its decomposition automatically, meaning the approach can use standard PDDL domains without additional agent markup. (See also (Nissim, Apsel, and Brafman 2012) for other work on automatically calculating decompositions.) Due to the structure of the problem domain, we could also ignore many of the complexities of multiagent planning, such as concurrency constraints for joint actions, which could instead be encoded encoded into the standard preconditions and effects of actions.

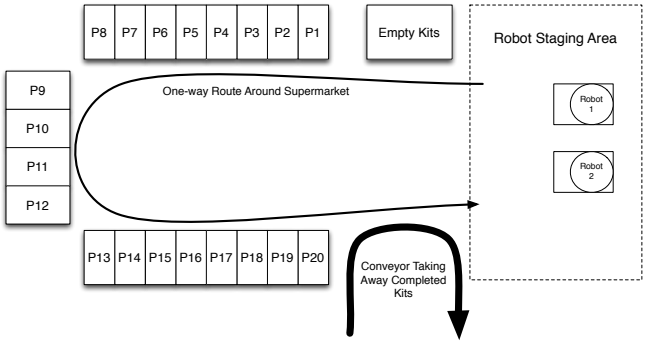

Figure 1: The warehouse planning environment. Robots must follow the track around the warehouse and cannot overtake. Parts are stored in labelled boxes, and must be appropriately selected to complete a kit. When complete, kits are placed on the conveyor. Robots may carry up to two kits at a time and may pass one another when in the staging area.

The major focus of this paper is to address an important problem in decompositional multiagent planning which is essential for planning in our domain: balancing the generated plans amongst the available agents. This problem is very similar to the issue discussed in (Borrajo 2013) in which different possible goal assignment strategies for multiagent planning are categorised. Borrajo proposed 4 different categories: *all-achievable*, *rest-achievable*, *best-cost* and *load-balance*. Of these categories, we found that the *all-achievable* decomposition gave the best results, and this appears in ADBP, albeit not explicitly. It's also worth noting that ADP uses a version of *best-cost*.

Problem Domain and Encoding

In this section, we briefly discuss the features of the problem domain and its encoding in PDDL. A pictorial representation of the domain is shown in Figure 1. In the domain, a fleet of robots must travel on a preset path and gather a given list of parts to assemble a kit. The robots cannot overtake each other except in the staging area (in which case they can return to their base and then leave before another robot). A kit box is a box with a number of compartments, each designed to fit a specific part. Each robot can carry up to two kit boxes at a time. The aim of the mission planner is to provide initial plans for each of the robots in the fleet so that they can complete their task of assembling kits within a specific time frame and without conflicting with other robots.

For the encoding, it was necessary to model the robots, the position of the kits on the robots, the kits themselves, the available parts, and the various locations in the warehouse. Robots then could perform the following actions: `move`, `pick-and-place`, `get-kit`, and `deliver-kit`. At first, the pick and place skills were modelled as separate actions,s which allowed the robot to travel while still holding a part in its gripper; however, this reduced the number of instances that the original planners tested could solve.[1]

[1]In practice, we found that we had to test a number of domain

Problem Number	No. of Robots	No. of Goals	POPF2			FF			LAMA			ADP		
			time(s)	cost	ms	time(s)	cost	ms	time(s)	cost	ms	time(s)	cost	ms
1	2	4	0.52	29	13	0.01	30	18	0	24	12	0.01	24	24
2	2	4	–	–	–	0.34	54	27	0.01	54	27	0	54	54
3	2	4	–	–	–	0.01	74	37	0.01	74	37	0.01	74	74
4	2	6	–	–	–	7.11	132	74	0.04	111	74	0.02	132	132
5	4	6	–	–	–	–	–	–	–	–	–	0.02	111	111
6	4	8	–	–	–	–	–	–	–	–	–	0.03	148	148
7	6	8	–	–	–	–	–	–	–	–	–	0.05	148	148
8	6	10	–	–	–	–	–	–	–	–	–	0.06	185	185
9	8	10	–	–	–	–	–	–	–	–	–	0.08	185	185
10	10	10	–	–	–	–	–	–	–	–	–	0.11	185	185

Table 1: Table showing the performance of planners on testing domains as the size of the problem increases to the size required for application in the real-world domain. Results show the time in seconds, the total cost and the makespan when the output is contracted to a multiagent plan with joint actions. A '–' means that the planner did not return a plan within 300 seconds.

One interesting aspect of the domain encoding concerned how to determine when a kit was full and ready to be placed on the conveyor. In the course of writing the PDDL files, multiple approaches were considered, such as:

- Numeric fluents that counted the number of parts in a kit and actions with conditional effects that considered whether a kit was full.
- A 'close-kit' action that could only be performed when the numeric fluents counted the correct number of parts in a kit. The 'close-kit' action would need to be performed before the kit could be deposited on the conveyor.
- Both of the above approaches, except with the numeric fluents represented by counting encoded in number objects with a successor relation.
- High-arity predicates or actions with a large number of parameters, coupled with either equality constraints (one for each possible pair of parts) to ensure that parts were not assigned twice, or a careful encoding to ensure that the parts were guaranteed to be different in any applicable action. For example, a high-arity 'deliver-kit' action requiring 6 parts might include the parameters:

```
(?r - robot ?l - location ?k - kit
    ?p1 - part ?p2 - part ?p3 - part
        ?p4 - part ?p5 - part ?p6 - part)
```

None of these attempts were solvable by the initial planners we tested for any but the smallest of domains. Even for the cases that were successful, the latter solutions were not well suited to problems with variable kit sizes.

In the end, we adopted a solution that encoded kit information as part of the goal for each problem, with separate propositions denoting 'delivered(kit)' and 'in-kit(part)'. This avoided the need for high-arity predicates or parameters, numeric fluents, and inequalities, and was the only method tested that allowed the temporal and single-agent planners to solve even small domain instances. However, as will be seen below, such an encoding created slight complications with the goal distribution during search in ADP.

Beyond the specific encoding choices, the particular domain we are encoding has many features that make it well suited to the approach presented in this paper. We list them here as a useful reference when considering similar possible applications of ADBP. In particular, the general features of our domain and its encoding include:

- A flat PDDL representation.
- Multiple agents that can act concurrently.
- No need to explicitly model concurrency constraints on joint actions.
- Goals that can be completed by multiple agents.

Initial Results

Using the above domain description, we tested our encoding on a set of off-the-shelf planners (POPF2, FF, LAMA, and ADP). The complete version of the domain was modelled with 10 robots, 10 goals and a kit size of 6, however, this problem was not solvable by most of the planners in our initial tests. We therefore created smaller problem instances for testing purposes by varying the number of agents and goals. Each problem still contained the full number of locations and parts, and parts-per-kit, except for two problems (0 and 1). These problems only required one part per kit, and one problem (0) included only a quarter of the number of locations and parts.

The results of the initial testing are shown in Table 1, and indicate that the traditional planning approaches start to break down as soon as the number of robots in the domain is increased. In some respects this could be regarded as counterintuitive, as adding robots in this deomain creates new possibilities for solutions to the problem, without removing any existing ones. However, without specific decomposition techniques, the increased size of the search space is clearly the dominant factor. While it would be possible to create more testing domains so that the behaviour of the algorithms can be discerned in more detail, it can be seen that the limiting factor is related to the number of agents and, once they are increased from 2 to 4, then only ADP can find solutions.

variations before we had an accurate encoding that was also solvable, even in the smallest instances, by the planners. We note that ADP could still find solutions when the actions were separated but there was no need to continue with this encoding for our purposes.

As well as time and cost, the results also show the makespan of the returned plans. For POPF2, this is simply the makespan as the temporal domain was designed with a direct mapping from actions at a time step in the temporal plan to joint actions. For the other approaches, the makespan can be calculated manually by post-processing plans using the algorithm presented in (Crosby, Jonsson, and Rovatsos 2014). For our particular domain, the highest number of actions assigned to a single agent is a good enough approximation of the final makespan, and there was no need to calculate the exact joint plan at this stage. We use this approximation in the results reported in this paper.

The makespan results show that, while ADP has no trouble finding solutions to the problem, the makespan exactly matches that of the full cost of the plan, meaning that each action is performed by a single robot. As all goals are achievable by all agents at the same initial estimated cost, this just happens to be the agent that is listed first in the internal representation of the planner.

Thus, while the initial testing demonstrated the utility of using a method like ADP in order to find solutions to our planning problem, it also illustrated the need to modify the ADP algorithm in order to return more balanced plans. This is the main focus of the work we describe below.

ADP Planning Formalism

The input to our planning formalism is a planning problem in typed PDDL, with no additional requirements necessary. The input is converted into an MPT $\Pi = \langle V, I, G, A \rangle$ (we assume there are no axioms in the domain) by the Fast Downward planning system (Helmert 2006), where:

- V is a finite set of state variables v, each with an associated finite domain D_v,
- I is a state over V called the initial state,
- G is a partial variable assignment over V called the goal, and
- A is a finite set of (MPT) actions over V.

A state is represented by a variable assignment, a function $f : V \rightarrow D_v$. An MPT action $a = \langle pre, \mathit{eff} \rangle$ consists of a precondition, a partial variable assignment pre over V, and a finite set of effects which are triples $\langle cond, v, d \rangle$, where $cond$ is a (possibly empty) partial variable assignment called the effect condition that must hold for the action to be applied, v is an affected variable, and $d \in D_v$ is the new value for v after completing the action. We say that $v \in pre(a)$ only if v belongs to the domain of the precondition of action a. We say that $v' \in \mathit{eff}(a)$ if v' is the v in some triple of the effects of a.

ADP Overview

The operation of ADP can be summarised by the high-level pseudo-code shown in Algorithm 1. The algorithm consists of an initial preprocessing agent decomposition phase, followed by greedy best-first search. States in the search (as well as being variable assignments over V) are associated with an agent, a set of goals, and a macro heuristic value, which are used to partition the problem into a single-agent subproblem for which the *hff* heuristic value is calculated (Hoffmann and Nebel 2001). The extra state information is carried over to successor states and recalculated at *coordination points*. A *coordination point* is any point where all the goals in $S.goals$ have been reached, the initial state, or a state in which it is not possible to reach $S.goals$ but that isn't a global dead end. (I.e., any point where the single-agent subproblem is no longer useful.)

Algorithm 1: High-level Summary of ADP

Input : MPT $\langle V, I, G, A \rangle$
Output: Plan or $\perp$
1 Calculate Agent Decomposition Φ
2 $S \leftarrow I$
3 Initialise $S.agent$, $S.goals$ and $S.macro$
4 **repeat**
5 Greedy BFS from S
[When the successor of S is generated it copies $S.agent$, $S.goals$ and $S.macro$ from its predecessor]
6 **until** *Goal Reached or Whole Space Explored*

ADP Decomposition

The first part of the ADP algorithm is a decomposition process by which the MPT is decomposed into multiple 'agents'. Given that the decomposition part of the algorithm was successful during our initial testing, we left this part unchanged, and do not provide exact details here. However, certain properties of the decomposition are relevant to understanding both ADP and ADBP, which we outline below.

Decomposition creates a partitioning of the variable set V into variable subsets ϕ_i for each agent i, and a public variable set P that contains the variables that pertain to the environment. The idea is that the elements of ϕ_i represent the internal state of an agent such that no other agent has access to an action that can change them. The elements of P represent the world that the agents are acting in, which can potentially be modified by any of the agents.

Actions in the domain are split amongst the agents based on the decomposition. An action is said to be an *internal action* of agent i iff:

$$\exists v \in pre(a) : v \in \phi_i, \text{ and}$$

$$v \in pre(a) \rightarrow v \in \phi_i \cup P.$$

In other words, the preconditions of a must depend on an internal state variable of i and can only change i's internal state variables or the domain's public variables.

A *public action* is any action where:

$$v \in pre(a) \rightarrow v \in P,$$

i.e., where the preconditions do not depend on the internal state of any of the agents. The decomposition algorithm only finds decompositions such that all actions become either internal or public, while trying to maximise the number of agents and minimise the number of public variables.

Algorithm 2: Heuristic Value Calculation of ADP

Input : State S with $S.agent$, $S.goals$ and $S.macro$
Output: $h + S.macro$

if *S is Coordination Point* **then**
  Relaxed Planning Graph Generation
  if *Max layer* > 0 **then**
    Calculate Subgoals
  Assign Goals
  $S.agent \leftarrow$ agent with most goals with min h_add
  $S.goals \leftarrow$ all goals achievable by $S.agent$
  $S.macro \leftarrow N \times |G \setminus S.goals|$
$h \leftarrow hff(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$

An agent's action set is the set of all internal actions of that agent, denoted by Act_i.

In a decomposition returned by ADP, the sets ϕ_i are guaranteed to have the *agent property*. In particular, a variable set ϕ_i (as part of a full decomposition Φ) has the agent property when for all $a \in A$ and variables $v \in V$:

$$v \in \phi_i \wedge v \in eff(a) \rightarrow a \in Act_i.$$

In other words, any agent variable can only be modified by an internal action of that agent. This corresponds to the idea that agent sets are sets of variables that represent the internal states of agents. For our domain, the decomposition separates out the variables for each robot corresponding to their locations, and the kits that they are currently holding.

Given this definition of a decomposition, we can create a taxonomy of the actions in the domain. For our purposes, the interesting types of actions are actions that are either *influenced* or *influencing*. An action is *influenced* (by the environment) if it contains in its preconditions a member of P; i.e., if another agent could potentially change whether or not it is applicable. An action is *influencing* (it influences the environment) if it contains in its effects a member of P; i.e., the action may change what another agent is capable of. An action can be neither *influenced* nor *influencing* if it only affects and depends on the internal state of an agent (e.g., a robot changing its gripper).

ADP Heuristic Calculation

The ADP heuristic calculation is formed in two steps. First, there is a global *coordination point* calculation that is performed infrequently and is used to pick out a single agent and a set of goals for that agent to attempt to achieve. If an agent has already been found and it still has achievable unmet goals then there is a local single-agent calculation of the FF heuristic value of the chosen agent. We first focus on the components of the *coordination point* calculation.

Relaxed Planning Graph Generation: This part of the coordination point calculation generates relaxed planning graphs for each agent. The information from these structures is then used to assign subgoals and to pick the agent for $S.agent$. Given a state, each agent generates their full relaxed planning graph for the restricted problem from that state. That is, each agent generates a graph containing all possible actions it can perform (ignoring delete effects) and all possible propositions that can be reached by performing those actions.

It may happen that some propositions can only be reached if agents cooperate with one another. For example, one agent may need to unlock a door before another can pass through. Because the agents create their own planning graphs (without interactions) they will not include any parts of the search space only reachable by cooperation. Therefore, if not all goals are reached in the initial calculation, the collected final state of all the agents is formed and used as input for a subsequent *layer* of relaxed planning graphs. Each successive iteration introduces a new *layer* with the first being *layer* 0. Repeating this process until no more states are added by any agent is guaranteed to cover every reachable state in the full problem.

Calculate Subgoals: If more than one layer of relaxed planning graphs have been generated, then subgoals are calculated. From our door example, if the second agent wants to pass through the door, it needs to assign the subgoal of unlocking the door to the first agent. Any time a goal proposition appears for the first time in a layer above 0 it cannot be reached by an agent on its own. Therefore, plan extraction is used to find out which proposition is utilised from the previous layer. All necessary propositions from layer 0 are added as subgoals to the agent that achieved them first. In our door example, the first agent will have the additional goal to unlock the door assigned to it.

Assign Goals to Agents: The next part of the coordination point calculation chooses which agent is going to be performing the local search. First, each goal is assigned to the agent that can achieve it with the lowest estimated cost from the relaxed planning graphs. That is, it is assigned to the agent that added it with the lowest h_add value. An agents goal set is then formed of all goals and subgoals assigned to it. The agent with the largest goal set is chosen as $S.agent$ along with all of its goals (and subgoals).

As a final part of the coordination point calculation the value $S.macro$ is calculated. This is used to provide a global heuristic estimate of how far through the search space we are. This is calculated as $N \times |G \setminus S.goals|$ where N is some large number chosen such that it dominates the FF heuristic value of a state.

Heuristic Calculation: Having had all the hard work already performed at the most recent coordination point, all that is left to do is calculate *hff* for $S.agent$ and $S.goals$. The heuristic value calculation is identical to that used by FF on the planning problem restricted to $S.agent$ and the returned heuristic value is added to $S.macro$. The idea is that (if we ignore backtracking) ADP essentially solves a series of single agent planning problems one after the other.

Applying ADP to the Problem Domain

We now discuss how ADP functions on our warehouse domain. We will be using output from problem 1 in Table 1. ADP finds the decomposition represented by ground instances of the following variables:

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
robot0	6	5	4	3	8	8	8	8
robot1	6	5	4	3	8	8	8	8

Table 2: The h_add values of each agent after generating relaxed planning graphs from the initial state of problem 1.

```
Agent0:
 ->at-robot(robot1, ?location)
 ->on-robot(robot1, ?kit, ?compartment)
 Agent1:
 ->at-robot(robot2, ?location)
 ->on-robot(robot2, ?kit, ?compartment)
```

In other words an agent is defined by its location and the kits that it is carrying.

The initial state is a *coordination point* so relaxed planning graphs are generated by each agent. A goal table can be output based on each agent's h_add value for each goal and is shown in Table 2. There are eight goals in problem 1, one for delivering each of the four kits and one for placing the correct part in each of the four kits. The first four numbers show the h_add value for putting a part in a kit and the latter four show the h_add value for delivering an empty kit. We can see that, from the initial state, each goal is achievable at the same estimated cost in each agents subproblems. There is no reason to choose one goal over another for assignment to an agent based on this information alone.

During the initial coordination point calculation no subgoals need to be calculated and ADP greedily assigns all goals to robot0. The search doesn't ever end up in a local dead end so single-agent search effectively continues until a plan is found. As a robot can only carry two kits at a time the same robot makes multiple trips around the warehouse in order to fulfil all the orders. This explains both the fast planning time of ADP and the high cost and makespan of the returned plans. This pattern is repeated in all the problem domains.

The ADBP Algorithm

In order to modify ADP for our purposes we need to take a less greedy approach to assigning goals to agents. However, as Table 2 starts to show, there is no obvious way to choose between goal assignments from the initial state. One reason for this is based on the choice to encode the information of which part should be put in which kit in the goal state itself. Referring back to the goal table for problem 1, it turns out that the goals g_1 and g_5 belong together as g_1 specifies the part that needs to be placed in kit1 and g_5 specifies that kit1 needs to be delivered. We attempted many methods for clustering these goals together but could not find a generalisable algorithm for doing so based on information available from the relaxed planning graphs.

The ADP paper claims that minimising the number of coordination points is important because the coordination point calculation is costly. However, we tested a version of ADP where an extra dummy coordination point calculation is performed at each state of the search. We found that the maximum time difference was on problem 10 of the testing set which took 0.25s instead of 0.11s. This means that, for our purposes, we can be more liberal with coordination point-like calculations. Unfortunately it also means that a lot of the power of ADP must be in the heuristic values it returns, which we have to change.

Algorithm 3: Heuristic Value Calculation of ADBP

Input : State S, Goals G
Output: h_max, h_square, or h_total

```
Relaxed Planning Graph Generation (full)
if Max layer > 0 then
    Calculate Subgoals
    G ← G ∪ subgoals
foreach agent i do
    foreach Goal g ∈ G do
        if h_add(g, i) > 0 then
            ExtractRelaxedPlan(g, i)
    hff(i) ← RelaxedPlan(i).cost
h_max = max_i hff(i)
h_square = Σ_i hff(i)^2
h_total = Σ_i hff(i)
```

The overall structure of ADBP is identical to ADP except that $S.agent$, $S.goals$ and $S.macro$ are no longer used. The new heuristic value calculation is shown in Algorithm 3. Compared to the original ADP the notion of coordination points is removed so that every state follows the exact same heuristic calculation. The Relaxed Plan Generation and Calculation of Subgoals remain, but goals are no longer assigned to agents and no single agent is chosen. Instead, each agent is included in the calculation of each heurisstic value. While this means that are algorithm performs slower than ADP, we found it necessary in order to output the kind of plans we were looking for.

The first step of the heuristic calculation is Relaxed Planning Graph Generation and this proceeds exactly the same as in ADP, except that we no longer stop if all goals have been reached by at least one agent. Instead generation is performed, layer by layer, until no further propositions can be added by any of the agents. This results in the full relaxed plan space being explored. The reason that we can no longer terminate relaxed planning generation early is that each agent that can achieve a goal (not just the first as in ADP) contributes to the heuristic value of a state.

The step to calculate subgoals is performed exactly as in ADP except that all found subgoals are added to the set of goals for the problem. These later form a part of the heuristic calculation from all agents and not just the agent that added it with the lowest cost as in ADP.

In the final step of the heuristic calculation each agent creates a relaxed plan to every goal in the goal set (including subgoals) that it can achieve. The value $hff(i)$ is the cost of the agent i's relaxed plan. This only counts each action once (even if it is used to reach multiple goal propositions). [2]

[2]Note that, like ADP, ADBP supports preferred operators. Any action that appears in a relaxed plan (of any agent) that is also applicable in the current state is set as a preferred operator. Without

Problem Number	No. of Agents	No. of Goals	Time (seconds)				Plan Length				Makespan			
			orig	max	squ	total	orig	max	squ	total	orig	max	squ	total
1	2	4	0.01	0.01	0.01	0.01	24	24	24	24	24	12	12	12
2	2	4	0	0.01	0.01	0.01	54	54	54	54	54	27	27	27
3	2	4	0.01	0.03	0.02	0.02	74	74	74	74	74	37	37	37
4	2	6	0.02	0.46	13.11	13.07	132	132	153	154	132	66	95	95
5	4	6	0.02	3.91	0.19	0.19	111	174	111	111	111	58	37	37
6	4	8	0.03	–	0.35	0.35	148	–	147	147	148	–	37	37
7	6	8	0.05	–	0.96	0.95	148	–	148	148	148	–	37	37
8	6	10	0.06	–	1.59	1.6	185	–	185	185	185	–	37	37
9	8	10	0.08	–	3.42	3.41	185	–	185	185	185	–	37	37
10	10	10	0.11	–	6.33	6.31	185	–	185	185	185	–	74	74

Table 3: Table showing the performance of the different versions of ADP on the testing domains. Key: orig is the original ADP algorithm. max is ADBP-max, squ is ADBP-square and total is ADBP-total.

Different Heuristic Values

We found that even slight variations on the previously explained heuristic calculation caused ADBP to perform much worse. However, there was some scope to test different possibilities for how the information in the relaxed plans was converted into a heuristic value. We tested three different versions:

ADBP-max uses the value h_max which is the maximum hff value of all the agents. This discards a lot of the information that has been calculated, but is as close as possible to the estimated makespan of the plan from the current state. As we are primarily concerned with minimising makespan this seemed like a natural heuristic value to investigate. The downsides of this calculation are that it discards a lot of potentially useful information and has little concern for how close we are to the goal state. For example, five agents with $hff = 9$ returns a worse heuristic value than if four have $hff = 0$ and one has $hff = 10$.

ADBP-total takes the other extreme and uses the sum of each agents hff values. This is much further from the makespan heuristic estimate but encodes more information about the distance from the state to the goal. It should be noted that this is different from the single-agent FF value for the state because goals are repeated by all agents that can achieve them and subgoals are included in the calculation. The downside of this calculation is that it does not take the makespan into account at all (beyond the fact the value is calculated over multiple agents).

ADBP-square attempts to sit in the space between the two extremes and uses the sum of squares of the hff values of the agents. The idea is to use all the information available whilst also taking the variance between the agent values into account.

Evaluation

Table 3 shows the results for the different versions of ADBP on the warehouse testing domains (the same problems as are shown in 1). The first thing to note is that the makespans of all the ADBP algorithms are significantly lower than those for the original ADP. All the versions achieve (to some extent) the goal of distributing the plans amongst the agents.

this both ADP and ADBP practically useless.

From the table we also see that ADBP-max is not able to solve any problem beyond number 5. However, this is still one more problem than both FF and LAMA were able to solve so there is still an improvement over the single-agent approach. This is perhaps somewhat surprising given the apparent lack of information contained in the heuristic value.

The most interesting feature about the results for ADBP-max is not shown in the table but can be found in the plans it outputs. Both ADBP-total and ADBP-square output plans in which the robots perform many actions in a row, normally completing a goal before another agent moves. Whereas the plans returned by ADBP-max contain actions that are very much interleaved between the different robots, the second agent follows directly behind the first as they navigate the warehouse. Given that we can compute a reduced makespan plan with post processing, this may not seem important, but the result shows an interesting area for future work where interleaved actions may be more significant.

The last conclusion to draw from Table 3 is that there was very little difference between ADP-square and ADP-total. We also note that both ADP-square and ADP-total find reduced makespan plans within our allotted 10 second time requirement. We hope to be able to update these results as we continue to work on the application, and study the behaviour of the domain and the algorithms in more detail.

Multiagent IPC Domains

We also tested the algorithms on a set of multiagent IPC domains (IPC 2011), to see if these techniques could be used to find reduced makespan plans there. The results of these tests are shown in Table 4. We did not expect to produce competitive times given that our version was performing sixty times slower than the original ADP in the largest warehouse domain instance. However, we did hope to show improved makespan over ADP and to be able to solve problems with a similar order of magnitude slowdown.

For the Rovers domain we can see that ADBP is not competitive with either the original ADP or traditional single-agent planners in terms of planning speed. As with the warehouse domain, ADBP-max cannot solve the larger problems. However, unlike in the warehouse domain, in this case ADBP is already failing on problems found trivial by existing planners. The makespan of the returned plans are not

Prob	Search Time (s)						Plan Length						Agent Makespan					
No.	ff	lama	a	a-m	a-s	a-t	ff	lama	a	a-m	a-s	a-t	ff	lama	a	a-m	a-s	a-t
Rovers																		
10	0.01	0.02	0.01	0.86	0.19	0.19	37	40	37	37	41	41	13	17	16	17	18	18
12	0	0.01	0	0.22	0.06	0.06	19	19	21	23	22	22	9	9	17	11	9	9
14	0.01	0.01	0.01	0.59	0.24	0.24	28	32	30	35	37	37	15	19	18	31	16	16
16	0.01	0.02	0.01	1.02	0.23	0.24	45	45	48	47	49	49	25	25	36	37	44	44
18	0.02	0.03	0.01	–	16.36	16.44	50	49	51	–	59	59	18	12	33	–	24	24
20	0.09	0.22	0.05	–	170	170.35	96	86	98	–	103	103	23	24	50	–	21	21
Satellite																		
10	0.01	0.04	0.02	–	3.58	3.58	35	39	51	–	43	43	14	18	47	–	19	19
12	0.03	0.09	0.04	–	13.91	13.9	45	44	56	–	54	54	24	22	46	–	26	26
14	0.05	0.14	0.04	–	6.87	6.63	43	45	55	–	51	51	18	29	41	–	23	23
16	0.05	0.13	0.05	–	59.51	59.09	49	52	50	–	65	65	31	19	33	–	19	19
18	0.02	0.12	0.02	–	6.77	6.91	38	44	41	–	48	48	23	14	13	–	16	16
20	0.04	0.92	0.09	–	85.3	84.95	107	113	115	–	114	114	36	30	97	–	38	38
Elevators																		
10	0.01	0.08	0.01	–	36.28	14.69	29	32	27	–	31	30	15	17	15	–	14	14
12	0.01	0.21	0.01	11.58	0.49	0.48	32	33	27	38	37	36	15	17	10	15	14	11
14	0.01	0.14	0.01	3.23	0.9	0.55	27	27	25	28	37	38	15	14	15	12	13	17
16	0.02	0.15	0.01	29.91	0.84	0.31	28	31	36	33	35	34	16	18	22	17	18	18
18	0.01	0.12	0.01	21.35	2.16	1.6	30	33	29	34	34	34	15	17	15	16	16	16
20	0.01	0.01	0.01	191.89	14.41	17.36	25	36	26	56	44	45	12	16	13	24	14	16

Table 4: Table showing the performance of different versions of ADP over IPC domains. Key: a is ADP, a-m is ADBP-max, a-s is ADBP-square and a-t is ADBP-total.

conclusively better than ADP's over the problems tested. We believe this is because agents in the Rovers domain have different capabilities, unlike in the warehouse domain, and not all goals can be completed by all agents. This is also indicated by the fact that ADP's makespans are low compared to its plan costs, even with it's completely greedy goal assignment strategy.

Satellite provides us with a domain where there is more choice in which agent solves each goal. Here we see ADBP having a much improved makespan over ADP in the majority of cases. However, the traditional single-agent approaches also find plans with low agent makespan. The planning times of the ADBP algorithms are again an issue, with ADBP-max not even managing to solve a single problem.

Finally, we tested ADBP on the Elevator domain as this domain had the property when used with the original ADP that it contained multiple layers and was therefore hard to solve. The Elevator problems used were those from the optimal track of the 2011 IPC competition (the problems from the satisficing track proving too complicated to show interesting results). These results finally show a divergence of the ADBP-square and ADBP-total versions of the algorithm, although this difference is still small. We believe that this could be in part due to using a greedy search algorithm, as ADBP-square and ADBP-total produce similarly ordered heuristic values when not comparing states that are very similar. The behaviour that ADBP-max exhibited in the warehouse domain is repeated in Rovers and Elevators. The returned plans contain a much stronger interleaving of different agent's actions than the plans of the other algorithms.

Conclusion and Future Work

In this paper we presented a new decomposition-based planning algorithm for use in a multi-robot mission planning application in a factory setting. We found that the proposed algorithm improved the quality of the returned plans compared to the existing algorithm, whilst still finding plans within our allotted time constraints.

We also tested our algorithm on multiagent IPC domains, but found that it is not competitive in its current instantiation. However, it is interesting to observe that ADBP works at all for domains with different properties to the warehouse domain. This indicates that the heuristic function is returning a somewhat useful estimate, even when the domain is not symmetrical and not all goals are completable by all agents.

The algorithm is currently at a state where it satisfies its intended role in our application domain, but will need to be improved and updated as the requirements of the environment evolve. It is expected that our encoding will be updated over time and it may be possible to find more efficient encodings, especially for use with decompositional approaches.

There is still much room for investigating different algorithms that lie between ADP and ADBP. For example, ADBP currently does not exploit the fact that some actions are not influencing and therefore cannot change the capabilities of the other agents in the system. After a non-influencing action is used to generate a successor state, there should be no need to preform the full heuristic calculation again.

Finally, if progress is made on the problem encoding or the planning algorithms, we intend to explore to what extent optimal planning can be employed. It seems likely it will be possible to find interesting pruning techniques based on the structure afforded by that decomposition of the domain.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreement no. 610917 (STAMINA).

References

Bøgh, S.; Nielsen, O. S.; Pedersen, M. R.; Krüger, V.; and Madsen, O. 2012. Does your Robot have Skills? In *Proceedings of the International Symposium on Robotics (ISR 2012)*.

Borrajo, D. 2013. Plan sharing for multi-agent planning. In *Proceedings of the Distributed and Multi-Agent Planning workshop*, 57–65. ICAPS.

Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 28–35.

Brenner, M. 2003. A Multiagent Planning Language. In *Proceedings of the Workshop on PDDL at the International Conference on Automated Planning and Scheduling (ICAPS 2003)*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 42–49.

Coles, A. J.; Coles, A.; Olaya, A. G.; Celorrio, S. J.; López, C. L.; Sanner, S.; and Yoon, S. 2012. A Survey of the Seventh International Planning Competition. *AI Magazine* 33(1):83–88.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *21st European Conference on Artificial Intelligence*.

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

IPC. 2011. http://www.plg.inf.uc3m.es/ipc2011-deterministic/. Web Site.

Nissim, R.; Apsel, U.; and Brafman, R. 2012. Tunneling and Decomposition-Based State Reduction for Optimal Planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 624–629.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)* 39:127–177.